Uses of Lightweight Containers and Operating System Virtualization

Jake Butler
CSU Stanislaus
408-348-7806
jbutler1@csustan.edu

ABSTRACT

A Container has the ability to not only provide a lightweight and scalable platform to test and develop software on, but provides several other benefits that can serve to advance industries such as software engineering and cloud storage. Various applications and uses for container based software is discussed within this paper.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Computing Platforms

Keywords

Containers, Virtualization, Docker, Virtual Machines, Software Engineering, Cybersecurity

1. INTRODUCTION

Virtual Machines have been the common software to provide some form of virtualization for whatever reasons an individual needs for that level of abstraction. However, VMs can be hindered by their limited scalability, and the fact that each instance of a VM requires a considerable amount of resources in order to run effectively. Containers offer a solution in the form of what can be considered a stripped-down version of a virtual machine. Each instance of a container provides all the software necessary to run a microservice, or an environment that can be dedicated to a single task.

2. What Are Containers?

According to the website of the popular container system Docker, a container can be defined as "a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings"("What Is A Container", 2017). Containers allow one to easily bundle an applications code, configurations and dependencies into manageable blocks that provide numerous benefits such as runtime consistency and version control. They utilize Linux cgroups and namespaces to maintain process isolation. Coincidentally, the very concept of containers was sprouted from trying to isolate namespaces in a linux environment. Processes appear to run on their very own system and provide a program with just enough storage and resources to run. This has led containers to be aptly named JeOS, or "Just enough OS".

At first glance, a container can appear to be very much similar to a virtual machine, in that they both provide a way to isolate applications and provide a virtual platform for applications to run on. However, the differences between the two highlight how resources are managed using containerization.

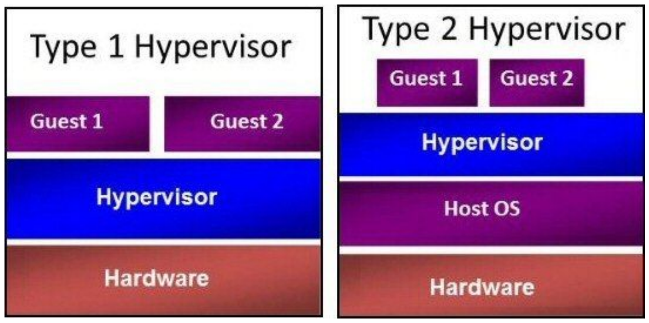


Figure 1: Difference between Type 1 and Type 2 Hypervisors

When a Virtual Machine is used, the VM uses what is called a "hypervisor" to run as an emulation layer. A hypervisor is a type of firmware that can either run directly off of the hardware (Type 1 Hypervisor) or off a hosting operating system (Type 2 Hypervisor). Without going too much into it, just know that hypervisors act as a layer of abstraction between the guest OS and whatever is underneath the hypervisor. While virtual machines do not need specific hardware in order to function, they are known to consume a considerable amount of storage and processing capacity compared to a server or a regular desktop.

The key difference between virtual machines and containers is the structure of the two technologies. With a virtual machine, each instance requires a guest operating system, along with whatever libraries, programs, and dependencies desired. Containers, on the other hand, do not require a guest operating system for each container. Rather, they can all be thought of as multiple user space instances that the kernel allows to run concurrently.

Container technology and virtualization began to formulate in the late 70's, when Unix-based systems utilized the *chroot* command, which allowed a user to change the root directory of a process and all of its children. This introduced a very early concept of process isolation and file segregation, and paved the way for early container technology. In early 2000's FreeBSD extended a virtual structure used by *chroot* called "jails", which allowed a systems administrator to partition a FreeBSD system into separate individual environments, complete with their own IP address and sets of configurations. Jails introduced a higher level of security, in that each jail was completely separate from the others, and it was not possible for a process from one to interact with the other, only with the main system. Later on in the 2000's saw the introduction of Solaris containers and OpenVZ technology.

Both are quite similar in implementation, with some stark contrasts. Namely that OpenVZ, which is run on most Linux distributions, limits resource usage per container image, whereas Solaris "zones" do not. These precursor technologies paved the way for LinuX Containers (LxC) which Docker and other services have branched their architectures from.

2.1 Container Architecture

The structure of a container itself can be separated into essentially four different layers. An advantage with containers, much like VMs, is that they can be run off of any OS, and this has even led to the creation of several open source projects to create specialized OSs for the sole purpose of running container technology off of them.

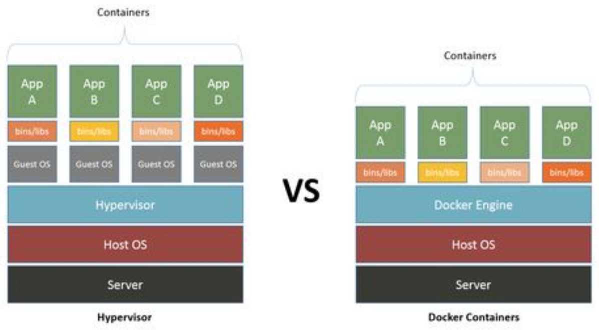


Figure 2: Difference between the structures of a Virtual Machine and the structures of a Container

As can be seen in the figure, containers operate quite similarly to virtual machines, with some key differences in their architecture. Each container comes equipped with its own set of libraries and dependencies that it runs with, which is supplied by the main engine that the containers run off of. The engines themselves client-server applications that consist of several layers to ensure that that the container images run efficiently. The Docker Engine will be used as an example for the following.

At the base level is the server, or the docker daemon, which is a program that runs as a background process rather than an interactive one. The docker daemon initializes and maintains docker images that are requested by the docker client. From there, a REST (Representational State Transfer) API is used to control the interactions that the client initiates to the daemon, which is all done through the Command Line Interface.

The Docker daemon is able to maintain all the images that run on the container instances using a registry that stores pre-built images. These registries can either be the public Docker Hub/Cloud or a private registry managed by an organization.

The containers themselves are given a layer of isolation through namespaces and control groups. Each container is given its own unique namespaces(i.e. a *pid* namespace for process isolation and a *net* namespace for managing the network interface.) Control groups are used to limit the amount of resources that are available to a certain container. This way, no container oversteps their boundaries and violates the isolation principle that builds the foundation of this technology.

3. Software Engineering and Research

In a truly perfect world, all softwares, from both commercial and research aspects, would be runnable on whatever platform an individual so chooses.
Unfortunately, in the real world, hardware and software are not as perfect as everyone would like them to be. Being able to replicate results from an experiment in a controlled and efficient environment, which is a core fundamental of all research, no matter the field, can be provided by using containers.

The paper *Reproducible Network Experiments Using Container-Based Emulation,* Nikhil Handigol outlines using container-based emulation to reproduce findings from research in networking systems.

3.1 Communication Networks

As previously mentioned, the ultimate goal of using containers for research is to be able to replicate any and all results from other papers and topics, so as to prove the validity of the thesis in question. Handigol highlights that the key ideals for a testing environment, dubbed "Mini-net Hifi", for networking systems research must be flexible enough to duplicate any topology needed, and at very little cost, in regards to resources and cost. He cites that, while other forms of testing, such as testbeds and simulators, are available, containers and container based emulators provide the best of both worlds while meeting all the criteria needed. Handigold states that "like testbeds, emulators run real code with interactive network traffic. [and] like simulators, they support arbitrary topologies, their virtual 'hardware' costs very little, and they can be 'shrink wrapped' with all of their code, configuration and data into disk images to run on commodity virtual or physical machines." (Handigold et al, 2012).

Mininet Hifi extends the functionality of the regular Mininet design by "adding mechanisms for performance isolation, resource provisioning, and monitoring for performance fidelity.[1] The scalability of container-based emulation is highlighted in the model built by the research team for this paper. Using a series of containers constructed through containers and linking the sessions through a series of Vswitches, the Mini-net Hifi model is able to effectively construct an emulator that mimics the structure of a basic topology. The containers act as a set of virtual hosts that are connected to a network namespace, which in turn "holds a virtual network interface, along with its associated data, including ARP caches and routing tables"(Handigold et al, 2012).

3.1.1 Results

The model constructed by Handigold and his team was given to eighteen groups of students with differing levels of computer expertise, with the purpose of having the latter recreate some finding that was published in a paper, and were given three weeks to do so. Out of these eighteen groups, "successfully reproduced at least one result from their chosen paper, [and] Four teams added new results, such as understanding the sensitivity of the result to a parameter not in the original paper."(Handigold et al, 2012).

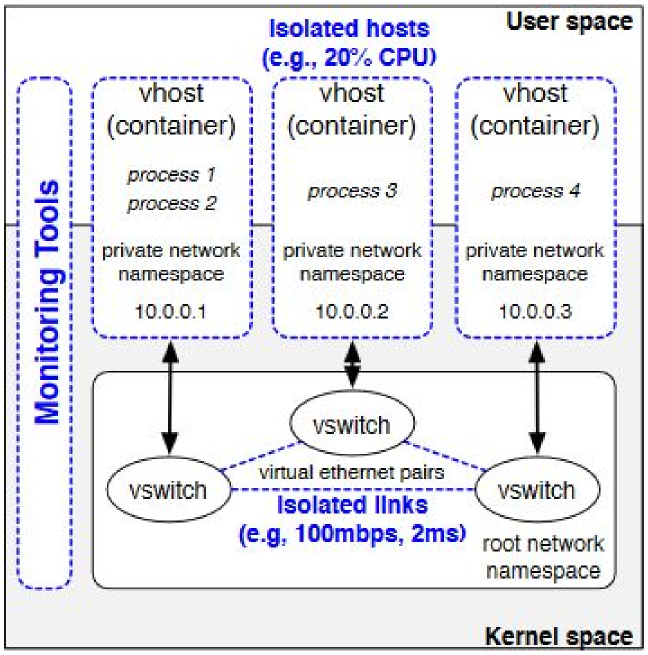


Figure 3: A series of containers act as virtual hosts to emulate a small network (Dashed lines indicate performance isolation)

The key concept to understand is that the use of containers for emulation made it simple to reproduce a wide range of experiments, demonstrating the versatility of container-based emulation. It is worth noting that while most of the experiments were successfully replicated, the degree of difficulty varied from group to group. Handigold notes that while projects centered around topics such as network configuration and data center networking went very well, others appeared to struggle with experiments that involved software that had not been patched or updated recently. The fact that the model also emulated a smaller network meant that there were difficulties if "the result depended on parameters whose dependence on link speed was not clear"(Handigold et al, 2012). Overall, the fact is that not only did majority of these groups successfully recreate network experiments inside a model that implemented container-based emulation, several of the groups were able to create more data that would adequately substantiate the original findings.

3.2 Data Analysis

It seems in today's day and age it is impossible to not hear the term "big data" in any industry. As businesses and corporations grow larger, so to does their need to access vast amounts of data and information in a timely and accurate manner. The same can be said for the field of research. However, the amount of data being generated is far outpacing the technology used to share and analyze these datasets. In *Container-based Analysis Environments for Low-Barrier Access to Research Data*, Craig Willis addresses this issue, and states that "there is a growing trend toward providing access to large-scale research datasets in-place via container-based analysis environments."(Willis et al, 2017). He goes further, and describes an approach that involves utilizing a platform known as the National Data Service (NDS) Labs Workbench, which in turn utilizes the Docker and Kubernetes container technology.

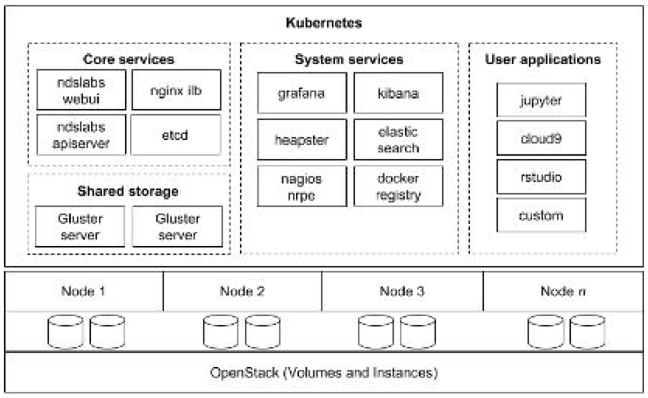
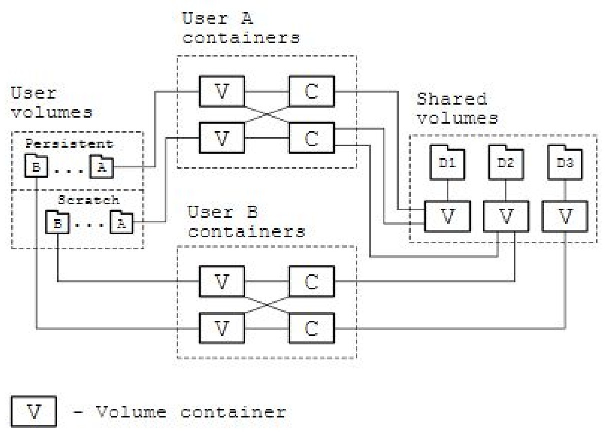


Figure 4: The architecture of the Labs Workbench platform

[1] The Original Mininet Model did not include any functionality to account for true performance isolation and limiting, so the HiFi model extended this using linux kernel cgroups and traffic control

Shown in the preceding picture is the structure that the Labs Workbench platform is developed into. The application uses container-based technology such as Kubernetes to deploy core services such as “the Nginx ingress load-balancer (nginx-ilb), which provides authenticated access to running containers in the cluster, as well as a thin REST API server (ndslabs-apiserver) an Angular user interface (ndslabs-webui)”, while Docker is used for image caching and running supported applications like Jupyter. While Kubernetes alone is not very efficient in terms of scalability, utilizing Docker in conjunction helps supplement what the former lacks.

Another platform mentioned is named SciServer Compute, which is discussed by Dmitry Medvedev in *SciServer Compute: Bringing Analysis Close to the Data***.** SciServer is described as “[a] big-data infrastructure project at Johns Hopkins University that is developing a modular and scalable infrastructure for the storage, access, query and processing of large, petabyte scale, scientific datasets.”(Medvedev et al, 2016). The application uses Docker containers to provide isolated workspace environments. Medvedev notes that a moderate weakness of Docker containers is the fact that “[they do] not easily scale across multiple nodes.”(Medvedev et al, 2016). As a way to address this issue, host nodes for Docker containers are run by virtual machines, so a new VM can be created and populated with more containers, in case a VM reaches the limit of containers that it can support given



its allocated resources.

The platform allows for users to create different preconfigured Docker container images, each that can come equipped with whatever libraries and dependencies are needed to perform the tasks. The images pull from three different kinds of file storage(which in of themselves are stored in Docker containers), which are all connected to the Docker cluster by default. As can be seen from the figure above, users pull whatever files from the storage system that is needed for their particular environment. This helps eliminate time spent waiting to download large volumes of data across networks, since all the necessary files are already connected to the user instance.

3.2.1 Summary

Figure 5: File System Organization for SciServer

As previously mentioned, the need for fast and efficient analysis of large amounts of data is a pressing matter. Cisco estimates that by the year 2021, IP traffic will “reach 278.1 Exabytes per month in 2021, up from 96.1 Exabytes per month in 2016”(Cisco, 2016), and the LHC at CERN alone produces up to 40TB of unfiltered data every second. Container-based analysis environments can provide lightweight and scalable platforms to access large amounts of research data, and an environment to replicate models that produced them.

4. Applications in the Cloud

According to the Cisco Global Cloud Index, the amount of total data stored in data centers will quintuple by 2020, from 171 to 915 exabytes, with traffic to and from data centers across the world reaching up to fifteen Zettabytes (or $1.5\,x\,10^{13}$ Giagbytes) in a year (Barnett et al., 2016). IDG’s Enterprise Cloud Computing Survey in 2016 revealed that over seventy percent of businesses have integrated Cloud storage into their business models, or plan on doing so. Needless to say, the demand for cloud services are on the rise for years to come. And with this comes along another slough of problems; namely being able to effectively distribute resources among jobs that may not require as much.

4.1 CoCOa

The Infrastructure-as-a-service (IaaS) model is a form of cloud computing model in which a cloud center provides components and resources to an organization that is usually present in an on-site data center. Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure are examples of independent IaaS services. However, the “pay-as-you-go” model that is implemented by IaaS services can be rather rigid in nature. Utilizing IaaS for smaller jobs and processes can leave a lot of unused resources that an organization still pays for, regardless of using it or not.

In order to better meet the demands of this workload, a system called Computing in Containers (Cocoa) is proposed by Ziaomeng Li and Fangming Liu in *Cocoa: Dynamic Container-Based Group Buying Strategies for Cloud Computing*. The term ‘Group buying’ is defined as a situation where “consumers enjoy a discounted group price if they form a group to purchase a deal [and] using price discount incentives, a cloud service provider can offer various group buying deals to attract users with complementary demands to form groups.” (Li et al, 2017). A certain number of parties agree to divvy resources amongst each other, provided the fact that none of them need more than they are given. Container-based technology helps to aid this strategy, due in part to the minimal overhead that containers incur and the flexibility that they offer.

Traditional workload consolidation utilized VMs, but small jobs pose challenges for this system. Namely, the dynamic nature of these small jobs means that VMs frequently had to cycle out jobs that have been completed, which in of itself can be a waste of resources. And the fact that small jobs may not utilize a great amount of CPU time may leave users paying for service time that they do not even use.

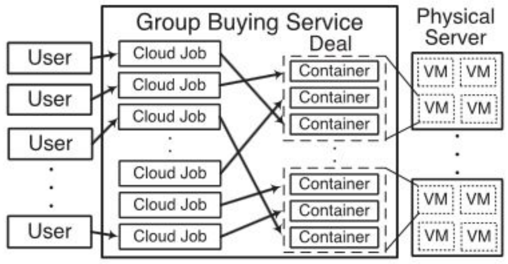


Figure 6: Framework of container-based group-buying service

Cocoa addresses this by dynamically grouping jobs together into batches, then allocating these jobs into groups of containers, which in turn are located in a specific VM. According to Li, after evaluating workload data collected from over ten thousand Google servers, “the performance of the approximation scheme proves to be close to optimal in our simulation”(Li et al., 2017). This container grouping can be achieved with native tools and APIs for container technology, such as Docker Swarm.

4.2 Grouping mechanism for CoCOa

In order to adequately allocate all jobs of various sizes, Cocoa needed to adapt both a static and dynamic strategy to group jobs together. Cocoa address this by using the static strategy to group newly assigned jobs into container batches. Static grouping is performed “[on] a batch of waiting jobs at the beginning of the group buying service, or whenever there is a sufficient number of jobs waiting to join buying groups” (Li et al., 2017). Cocoa must efficiently divide a vector of computing resources (defined by the paper as $Cj = (c1,\ c2 \ldots cN)$ among a group of *m* jobs. The group of jobs is assigned into an instance of a VM, which assigns containers to each job. Li defines the static group problem as a “in packing problem in which the target is to pack a set of variable-sized items into a number of bins at a minimal cost [and] we consider group buying deals as bins and user demands as items”(Li et al, 2017). To address dynamic grouping, Cocoa maintains a queue of all the running groups that checks at regular intervals. Containers are eliminated once the jobs contained within them are complete, and a group is removed. The advantage in this is that containers can be easily created and destroyed with very minimal overhead and setup time. The takeaway here is that the goal of this optimization model is to minimize the cost of the group buying, and in a manner that satisfies all the demands of the users in the group. Containers are able to achieve this due in part to their flexible nature.

$$
\begin{aligned}
\min \quad & \sum_{1\leqslant j\leqslant n}\sum_{1\leqslant s\leqslant m} p_j y_{js} \\
\text{s.t.} \quad & \sum_{1\leqslant j\leqslant n}\sum_{1\leqslant s\leqslant m} x_i^{js} = 1, \quad i \in \{1,\ldots,m\}, \\
& \sum_{1\leqslant i\leqslant m} r_{ik} x_i^{js} \leqslant c_{kj} y_{js}, \\
& j \in \{1,\ldots,n\},\ s \in \{1,\ldots,m\},\ k \in \{1,\ldots,d\}, \\
& x_i^{js} \in \{0,1\}, \\
& j \in \{1,\ldots,n\},\ i \in \{1,\ldots,m\},\ s \in \{1,\ldots,m\}, \\
& y_{js} \in \{0,1\}, \quad j \in \{1,\ldots,n\},\ s \in \{1,\ldots,m\}.
\end{aligned}
$$

Figure 7: Static grouping strategy presented as a five stage optimization problem

4.3 Summary

The entire concept of cloud technology itself is based on some level of virtualization. While VMs have been used due to the fact that they utilize hardware more effectively, containers are able to scale better and handle more jobs while consuming less resources. Virtual infrastructure and architecture is what drives the cloud centers and the IaaS model, and it is absolutely necessary to stay in pace with the increase of data being stored in the cloud. Containers provide many potential advancements in cloud storage, whether used on their own or in conjunction with some other virtualization technology. Platforms such as Youtube, Google Search and more have been developed and deployed through use of containers, with many more platforms beginning to incorporate the technology.

5. Conclusion

In this paper, I discussed containerization technology and several of the fields that it is being applied to. Containers give developers the ability to isolate their applications in a flexible and minimal environment that can be deployed on practically any system. From scientific research to cloud deployment, the container systems have started to become more integrated into numerous uses for different reasons. I predict that in the future, the technology of containers will diversify to each field over time. Both Docker and Kubernetes themselves are FOSS (Free and Open Source Software), which themselves branched from LxC. Different communities will use them for different needs, eventually adapting unique forms of container engines. I also believe that containers will lead more organizations to adopt cloud technologies, either to centralize all their data and resources, or to host legacy software. A quote from software engineer Marc Andreessen states that “Software is eating the world.”. If that is the case, then containers will be the table that it is served on.

6. **REFERENCES**

[1] What Is A Container?(n.d.) Retrieved from https://www.docker.com/what-container

[2] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., & Mckeown, N. (2012). Reproducible network experiments using container-based emulation. *Proceedings of the 8th international conference on Emerging networking experiments and technologies - CoNEXT 12*. doi:10.1145/2413176.2413206

[3] Willis, C., Lambert, M., Mchenry, K., & Kirkpatrick, C. (2017). Container-based Analysis Environments for Low-Barrier Access to Research Data. *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact - PEARC17*. doi:10.1145/3093338.3104164

[4] Medvedev, D., Lemson, G., & Rippin, M. (2016). SciServer Compute. *Proceedings of the 28th International Conference on Scientific and Statistical Database Management - SSDBM 16*. doi:10.1145/2949689.2949700

[5] Barnett, Thomas, et al. "Cisco Global Cloud Index." *Cisco Global Cloud Index*, Cisco, 15 Nov. 2016, www.cisco.com/c/dam/m/en_us/service-provider/ciscoknowledgenetwork/files/622_11_15-16-Cisco_GCI_CKN_2015-2020_AMER_EMEAR_NOV2016.pdf.

[6] Yi, X., Liu, F., Niu, D., Jin, H., & Lui, J. C. (2017). Cocoa. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems, 2*(2), 1-31. doi:10.1145/3022876